10/829,613 PTO-892

PCT WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 6 : G06F 17/30

A1

(11) International Publication Number: **WO 99/48029**

(43) International Publication Date: 23 September 1999 (23.09.99)

(21) International Application Number: PCT/US99/05716

(22) International Filing Date: 16 March 1999 (16.03.99)

(30) Priority Data:
09/039,728 16 March 1998 (16.03.98) US

(71) Applicant: MICROSOFT CORPORATION [US/US]; One Microsoft Way, Redmond, WA 98052-6399 (US).

(72) Inventors: GRAEFE, Goetz; 2820 169th Avenue S.E., Bellevue, WA 98008-5634 (US). ALGER, Jeff; 24066 N.E. 53 rd Place, Redmond, WA 98053 (US).

(74) Agent: VIKSNINS, Ann, S.; Schwegman, Lundberg, Woessner & Kluth, P.O. Box 2938, Minneapolis, MN 55402 (US).

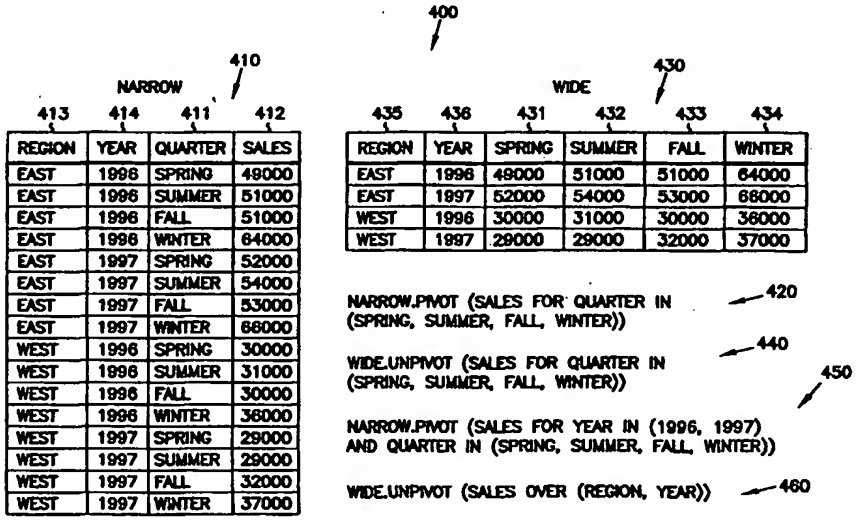
(81) Designated States: AE, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, CA, CH, CN, CU, CZ, DE, DK, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MD, MG, MK, MN, MW, MX, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, UA, UG, UZ, VN, YU, ZA, ZW, ARIPO patent (GH, GM, KE, LS, MW, SD, SL, SZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG).

Published

With international search report.

Before the expiration of the time limit for amending the claims and to be republished in the event of the receipt of amendments.

(54) Title: PERSPECTIVE TRANSFORMATIONS ON RELATIONAL DATABASE TABLES



(57) Abstract

A "pivot" operation rotates the data items in a relational database table so that certain data values in the table become column names of the pivoted table, and the data items of a specified value column appear in corresponding rows in the new columns of the pivoted table. A pivot list specifies that only certain values of the pivot column data items participate in the operation. Additional columns of the input table appear as columns in the output table; the rows of the output table are grouped by equal data-item values in these grouping columns. An "unpivot" operation provides the inverse of the pivot operation. Both operations may be nested in an SQL user query at the algebraic level. The operations occur in the search engine of a relational database management system, and may also be invoked as part of an optimization of another query.

FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AL	Albania	ES	Spain	LS	Lesotho	SI	Slovenia
AM	Armenia	FI	Finland	LT	Lithuania	SK	Slovakia
AT	Austria	FR	France	LU	Luxembourg	SN	Senegal
AU	Australia	GA	Gabon	LV	Latvia	SZ	Swaziland
AZ	Azerbaijan	GB	United Kingdom	MC	Monaco	TD	Chad
BA	Bosnia and Herzegovina	GE	Georgia	MD	Republic of Moldova	TG	Togo
BB	Barbados	GH	Ghana	MG	Madagascar	TJ	Tajikistan
BE	Belgium	GN	Guinea	MK	The former Yugoslav Republic of Macedonia	TM	Turkmenistan
BF	Burkina Faso	GR	Greece			TR	Turkey
BG	Bulgaria	HU	Hungary	ML	Mali	TT	Trinidad and Tobago
BJ	Benin	IE	Ireland	MN	Mongolia	UA	Ukraine
BR	Brazil	IL	Israel	MR	Mauritania	UG	Uganda
BY	Belarus	IS	Iceland	MW	Malawi	US	United States of America
CA	Canada	IT	Italy	MX	Mexico	UZ	Uzbekistan
CF	Central African Republic	JP	Japan	NE	Niger	VN	Viet Nam
CG	Congo	KE	Kenya	NL	Netherlands	YU	Yugoslavia
CH	Switzerland	KG	Kyrgyzstan	NO	Norway	ZW	Zimbabwe
CI	Côte d'Ivoire	KP	Democratic People's Republic of Korea	NZ	New Zealand		
CM	Cameroon			PL	Poland		
CN	China	KR	Republic of Korea	PT	Portugal		
CU	Cuba	KZ	Kazakstan	RO	Romania		
CZ	Czech Republic	LC	Saint Lucia	RU	Russian Federation		
DE	Germany	LI	Liechtenstein	SD	Sudan		
DK	Denmark	LK	Sri Lanka	SE	Sweden		
EE	Estonia	LR	Liberia	SG	Singapore		

PERSPECTIVE TRANSFORMATIONS ON RELATIONAL DATABASE TABLES

Background of the Invention

The present invention relates to electronic data processing, and more specifically concerns new query operations for the manipulation of tables in relational databases and similar types of data-management software.

A database is a collection of data in an organized structure. A typical database is stored in a computer as a set of records each having a number of fields for holding data items of a particular kind, such as character strings, numbers, or pointers to data located somewhere else. A relational database comprises any number of rectangular tables. Each table has a set of records; each record is referred to as a row of its table. Each record in the same table has the same number of fields. (However, some fields in a record may hold no data, indicated by a NULL value.) The fields of a table form a set of columns, which may have designated names that are not part of the data itself. The records do not have external to identify them individually. Instead, they are accessed by a key consisting of the contents of some combination of the fields; that is, a relational database may be considered to be a software-implemented content-addressable memory.

A database management system (DBMS, or database system) is computer software for storing, maintaining, and searching the data in a database. A DBMS usually includes facilities for increasing performance, reliability, and integrity, such as indexes, logging, and record locking. It always includes one or more interfaces for finding particular data from the database and for presenting these queries to a search engine. The engine searches the database and returns to the user a result, usually in the form of a relational table, which matches the specifications of the query.

The most widespread interface for relational databases is Structured Query Language (SQL). Although many variants of this interface language exist, standard versions have been defined by the American National Standards Institute (ANSI) and the International Standards Organization (ISO). Most

present commercial realizations of SQL follow these standard versions, although many of them include language constructs in addition to those defined in the standard, or at different levels of compliance.

Relational databases and relational query languages treat data as a set of rectangular tables. Yet many databases are conceptually multidimensional, based upon axes such as time {day, month, year}, locale {store, city, state}, category {product, product_group}, actor {clerk, department, division}, payment {cash, check, credit}, and so forth. A user often finds it useful to think of such data as a collection of collections, and may wish to view them from different perspectives. In the above example, one perspective is a collection of records, where each record represents a locale, and contains a collection of monthly sales data for that locale; another perspective sees a collection of records (i.e., rows of a table) where each denotes a particular point in time, and the fields of each record (i.e., the columns of the table) collect sales figures for the different categories.

From this point of view, the ability to transform a database table from one perspective to another—to rotate the dimensions of the data—would be a valuable addition to the conventional capabilities of a query language such as SQL. In this context, to rotate perspectives or dimensions means to interchange a dimension represented in a table as a set of columns with a dimension represented as a set of rows. Conventional relational DBMS products and standards offer no direct operation for rotating perspectives. Although it is possible to formulate SQL queries to achieve this effect indirectly, such queries are large, complex, error-prone, slow, and hard to optimize into efficient execution plans, even when parallel processing is available.

Some conventional spreadsheet software allows a user to interchange data in a user-selected rectangle of cells to be interchanged in the same way that a matrix-algebra "transpose" operation relocates a matrix element a_{ij} to a_{ji}. In the Microsoft Excel product's Pivot Table feature, for example, a user selects a rectangle of cells, copies it into a temporary clipboard, points to a destination cell, and performs a "paste special" operation after selecting "transpose" from an options menu. With a suite of compatible application programs such as Microsoft Office, a user may even select data from a database table in the Microsoft Access database component, transfer it as a single object to the Excel component as a rectangle

of spreadsheet cells, transpose the cells, then transfer the cells back into the Access database as a collection of records in the transposed format.

Transposing data items in this manner is both clumsy and functionally limited. Even for small databases, the invocation of another application program merely to carry out a single query is wasteful. For large databases, the conventional requirement that transposed data reside in memory renders this method impossible. For client/server architectures using host-based search engines, there is no way to connect to a spreadsheet program for performing the operation. In any environment, transposition via spreadsheet requires manual intervention, and thus does not permit a transposition to form an internal part of a query within a database program. Such external operations cannot participate in the sophisticated reformulation, rewriting, and other optimization procedures of conventional database-query processors and other search engines. On a more conceptual level, fundamental differences between spreadsheets and relational database tables prohibit the desired types of transposition. For example, the names of the columns or fields in a database table are not a part of the table itself; they do not form a record of the table in the way that column headings in a spreadsheet are a row of cells within the spreadsheet. Transposing a rectangle of cells in a spreadsheet thus cannot transform a column of cells into the names of columns when the spreadsheet rows return to the database program as records in a table.

The execution engine of the Microsoft SQL Server product has a strictly internal operation for splitting each item of a table update having the form `(row_identifier, old_values, new_values)` within a stream of update items into a "delete item" and an "insert item" which interchanges certain row and column values, and a similar operation for collapsing a "delete item" and an "insert item" into an "update item". These operations are not available to users and cannot participate in user queries. That is, the query processor uses them internally only for facilitating the efficient execution of certain functions performed while updating databases.

Thus, the database art could be significantly expanded by providing a facility for fast, efficient rotation of perspectives, especially for relational databases. Moreover, there is a need for rotation or transposition operations whose semantics and syntax integrate well into query languages such as SQL as

natural extensions, and which can be optimized and executed in conventionally organized database query processors and other search engines without adding complex or idiosyncratic facilities.

Summary of the Invention

The present invention provides a "pivot" operation for transforming the rows (records) and columns (fields) of a table, as that term is defined in a relational database, so as to provide different perspectives into the data items in the table. The operation accepts an input table and a pivot specification, and produces an output table. It takes place in the interface-language organization in such a way that it can be easily integrated into conventional database query processors, search engines, and servers. The operation places data in the fields of specified table records into the same field of different records, using the values of one or more designated table column as the names of the fields themselves. Data in any further columns are grouped by data values in a pivoted table.

It is sometimes easier to perform other relational operations upon a database table from another perspective, even when the ultimate result will have the original perspective. Therefore, the invention also provides an "unpivot" operation as an inverse to the pivot facility. Also, sometimes it is desirable to unpivot a stored table or intermediate result.

These operations, along with a simple and intuitive way of incorporating them into database queries, simplifies the writing of queries and makes them less error-prone. For example, they reduce or eliminate the need for joining tables to themselves. The method of invoking the operations permits deep nesting of multiple operations with a simple and powerful syntax extension and well-defined semantics, and applies a familiar programming-language paradigm. Permitting text as method arguments in queries enhances the power and ease of use of the extended SQL language. Moreover, expanding the set of relational-algebra expressions available in this manner to nonprocedural query expressions may also be applied to other operations, such as `sample`, `top`, and `rank`.

Pivot and unpivot operations according to the invention are inherently compatible with many types of data-manipulation software, and system architectures, especially including relational databases. These operations can be integrated into such systems both at the language level (e.g., by means of intuitive extensions to SQL and other query languages) and at the processing level (e.g., query optimization and execution).

Integrating data from multiple databases into a single data-warehouse database frequently faces an "impedance mismatch" when the multiple data sources have mutually differing shapes or row/column ratios. Almost by definition, such databases can be extremely large. Normalizing such data may depend upon context: storing data in pivoted form or perspective may be optimal—or even necessary—for one schema, while another schema may prefer or require the unpivoted form. Therefore, adding pivot and unpivot operations can greatly benefit the combination of data from different sources, especially large amounts of data.

The new operations provided by the invention also expedite lower-level DBMS processing, even with limited system resources. The extensible syntax and clear semantics of the new operations facilitates automatic generation and optimization of complex queries, especially in the rewriting of queries for more efficient execution. Even purely internal DBMS functions, such as update processing for index and integrity maintenance and other purposes, can benefit. The processing of SQL queries involving `IN`, `OR`, and `UNION` queries can be enhanced. Many optimization techniques already employed for `GROUP BY` queries are routinely adaptable to processing pivot and unpivot queries. Conventional execution algorithms including parallel-processing techniques for these queries apply to pivoting tables or query results, including unsorted and partitioned tables and results.

Other features and advantages of the invention, as well as variations within the scope of the invention, will appear to those skilled in the art from the following detailed description.

Brief Description of the Drawing

Fig. 1 is a block diagram of a computer network environment for the invention.

Fig. 2 is a diagram of a database management system for hosting the invention.

Fig. 3 is a flowchart of the functions performed by the DBMS of Fig. 2.

Fig. 4 illustrates examples of pivot and unpivot operations according to the invention.

Fig. 5 is a flowchart of a pivot operation according to the invention.

Fig. 6 is a flowchart of an unpivot operation.

Detailed Description

Exemplary Operating Environment

Datanase management systems are implemented in many different types of data-processing systems, including standalone personal computers, midrange and mainframe computers, peer-to-peer and client/server networks, and wide-area distributed systems of many architectures. All data-processing systems are suitable environments for the present invention. For purposes of exposition, however, the invention will be described in connection with a conventional client/server computer system 100 , shown in Fig. 1. Network wiring 110 interconnects a number of personal computers (PCs) 120 to a server 130 via network adapters 121 and 131. Server 130 includes a storage subsystem 132 for holding the large amounts of data in typical enterprise databases. Other system architectures are also suitable environments for the invention; for example, units 120 may be terminals connected to a mainframe or midrange computer 130, or unit 130 may itself comprise a PC coupled to PCs 120 in a peer-to-peer network. For small and modest databases, the entire system 100 may comprise a single PC acting as both client and server. Likewise, file storage may be distributed among a number of different machines. Fig. 1 shows schematic representations of an external storage medium 133 which may store client and server software

for distribution and downloading to clients, and another medium 134, such as a diskette, for offline storage of database tables.

Figure 1A and the following discussion are intended to provide a brief, general description of a personal computer 120. Although not required, the invention will be described in the general context of computer-executable instructions, such as program modules, being executed by a personal computer. Generally, program modules include routines, programs, objects, components, data structures, etc. that perform particular tasks or implement particular abstract data types. Moreover, those skilled in the art will appreciate that the invention may be practiced with other computer system configurations, including hand-held devices, multiprocessor systems, microprocessor-based or programmable consumer electronics, network PCs, minicomputers, mainframe computes, and the like. The invention may also be practiced in distributed computing environments where tasks are performed by remote processing devices that are linked through a communications network. In a distributed computing environment, program modules may be located in both local and remote memory storage devices.

Fig. 2 is a block diagram of a typical conventional client/server database management system 200 capable of operating in system 100, Fig. 2. A client application program 210 executes within each PC 120, under a PC operating system 220 such as Microsoft Windows95. Among other functions, client application 210 contains a facility 211 for accepting database queries from a user at a PC 120. In addition to user entries, other application programs 230 executing in some of the PCs 120 may present queries to DBMS client 210, via predefined host-language application-program interfaces (APIs) 231.

Within server 130, a DBMS server application 240, such as Microsoft SQL Server, executes under a server operating system 250 such as Microsoft Windows NT. DBMS program 240 provides services for creating, querying, maintaining, and modifying a number of relational databases, exemplified by database 260. Program 240 may employ the file-system services 251 of operating system 250, or may provide its own file system. Operating system 250 could execute a separate instance of the entire DBMS application for each request from a client 210. For greater efficiency, however, program 240 gives each client connection a separate thread 242 in the DBMS kernel. Further, this thread may be a native

operating-system thread, which carries with it all the Windows NT mechanisms for process memory protection, better access to storage devices, and so forth. Search engine 241 processes queries and other requests from individual clients 210 upon tables 261 of a database 260, as described more fully below. It also enforces database integrity with conventional facilities for record locking, atomic transactions, etc. In Microsoft SQL Server, the interface language between query facility 211 and search engine 241 is Transact-SQL, which provides most of the functions of the standard ANSI SQL 89 and ANSI SQL 92 languages, plus extensions for providing greater flexibility and programmability.

Fig. 3 illustrates some conventional functions 300 of search engine 241, Fig. 2, for processing a query transmitted from any of the client applications 210. SQL is a nonprocedural language because an SQL query is a specification of properties or predicates of a desired result, rather than a sequence of steps for obtaining the result. That is, a query such as `SELECT year, quarter, sales FROM Narrow WHERE sales < (SELECT AVG(sales) FROM Narrow) ORDER BY year, quarter` specifies the properties of an output table. The columns of the output table correspond to the columns named `year`, `quarter`, and `sales` taken from an input table named `Narrow`. The output-table rows (records) are to be ordered (i.e., sorted) by `year` and then by `quarter` within each `year` value. The records from the input table which appear in the output are only those where the value of `sales` is less than the average value of all values of `sales` in the table named `Narrow`. The nested subquery `SELECT AVG(sales) FROM Narrow` generates a table having only a single column and a single row containing the average value of `sales` in the `Narrow` table. The manner and sequence in which the records of the input table are accessed, and other details of the procedure or plan for building the output table are not defined by the query itself.

When search engine 241 receives a query, it parses the query into an internal or tokenized form, as shown in step 310. Validation step 320 ensures that the data named in the query actually exists in the database, and checks data and integrity constraints. It may expand some parts of the query, such as macros and views, at 321. Output 322 reports any errors back to the user or other source of the query.

All but the most limited search engines perform extensive optimization upon the query, as indicated in step 330. Optimization may involve rewriting the query by combining or splitting portions of the query, rearranging operations and subqueries, etc., and other methods, such as sequencing accesses to the records of stored database tables, and modifying functions For each candidate execution strategy, they calculate a cost value representing the computing time or resources required to execute the query using this strategy, and then select one strategy among all the possible candidates. Although the art of designing these optimizers is complex and arcane, those adept in it have the ability to adapt conventional optimizers so as to include new query functions of various types; designers of translators for other, more procedural languages also routinely construct optimizers of this same general class. A survey paper, M. Jarke and J. Koch, "Query Optimization in Database Systems," *ACM Computing Surveys* 16, 2 (June 1984), p. 111, discusses the construction of database query optimizers in more detail.

The output of step 330 is a query evaluation plan (or simply "plan") for answering the query. Step 340 compiles this plan into a procedural form, usually represented as a conventional function tree. Step 350 may then run a simple tree-traversal algorithm for executing the plan against the database objects. The output of step 350 is the result of the query, in the form of an output table returned to the source of the query. Search engines other than the one described herein may combine or divide the individual steps 300, or may omit or add steps. Another survey paper, G. Graefe, "Query Evaluation Techniques for Large Databases," *ACM Computing Surveys* 25, 2 (June 1993), p. 73, hereby incorporated by reference, addresses the subject of query execution, and cites a number of references providing additional description and discussion. Again, the steps of the newest search engines are specifically designed for easy extensibility to accommodate new syntax, new query function, optimization knowledge, and execution technology.

Pivot and Unpivot Operations

Fig. 4 shows the structure of a pivot operation according to the invention. This operation fits into the hierarchy of SQL operations at the level of relational algebra. Those who design relational database

systems and interfaces divide query processing into three levels. Because the mathematical theory of relations provides the conceptual framework for this type of databases, the first and second levels are frequently called the relational calculus and the relational algebra.

The relational calculus, like any calculus, deals with a high-level description or specification of a desired result, without naming any operations, procedures, or other method for obtaining the result. That is, it merely expresses the definition of a desired result relation (table) in terms of existing relations in a database. The query `SELECT employee.name, department.name FROM employee, department WHERE employee.dept_id = department.dept_id`, for example, describes the properties and constraints of an output table in terms of one or more input tables in terms of a typical member of the result relation and a qualification representing the defining property of the result members. The relational calculus provides the foundation for a formal, exact understanding of databases, tables ("relations"), and queries, and has found a commercial realization in the query components of the database language SQL, now an ANSI/ISO standard. Given the important role SQL plays in data-management products, extending database functionality for the real world requires that any added functionality should become a syntactically and semantically clean extension of the SQL language.

Relational algebra is more operationally oriented (yet equivalent to) relational calculus. Operations or functions in relational algebra consume one or more input tables and produce an output table according to a rule. For example, the relational operation `JOIN [employee.dept_id = department.dept_id] (employee, department)` combines the tables named `employee` and `department` along a common column or field named `dept_id` in both tables. (This is analogous to an operation such as addition, which consumes two numbers and produces a third, as, for example, the operation "4+5" produces "9".) Key characteristics of relational algebra are that: (1) operations consume and produce objects of the same type, namely relations; (2) operations can be nested into arbitrarily complex structures; and (3) new operations can be added. In the relational algebra, input objects not only have inputs, but may also carry tags denoting additional information. In the example immediately above,

the `join` operation not only specifies the two relational-algebra expressions, namely the two tables `(employee, department)` to be joined, but also names a "join predicate" specifying how they are to be joined: along equal values of a particular column in each table, `[employee.dept_id = department.dept_id]`.

Some very useful query operations are difficult to express at the relational-calculus level, yet they integrate easily and cleanly into the relational-algebra level. For example, `OUTER JOIN`, a variant of the relational `JOIN` operation, does not fit easily or cleanly into the simple `SELECT ... FROM ... WHERE` query syntax. Therefore, ANSI/ISO permit a limited set of relational-algebra expressions in place of tables in the `from` clause, e.g., `SELECT employee.name department.name FROM employee LEFT OUTER JOIN department ON employee.dept_id = department.dept_id`. That is, there is a precedent for extending a relational-calculus query with a relational-algebra expression, although such extensions have been thus far restricted to variations of `JOIN` operations. Relational-algebra operations often participate in the optimization of queries having selections, projections, aggregations, and other nonprocedural specifications at the relational-calculus level, as in block 330, Fig. 3.

Query-execution plans constitute the third and lowest level of query processing. Although the nesting of relational-algebra operations may indicate an order of execution, algorithms or sets of particular instructions for producing intermediate results occur at the level of execution plans, rather than at the higher levels. For example, there are three basic methods for performing relational `JOIN` operations: nested loops, merge-join, and hash-join; and each method has a large number of variants. Execution plans clearly indicate the choice among such alternatives, and are formulated at the lowest level of query processing in block 340, Fig. 3.

Because relational query processing is defined very precisely and within a definite structural framework, it is important to define any new functionality at all three levels: language extensions, relational-algebra operations, and execution plans. The invention may provide the pivot and unpivot

functions as new relational-algebra operations that participate explicitly in SQL queries, as extensions to the language.

The formal definition of a pivot operation, for an input `table-expression` in first normal form that is a valid query expression, is: `Table.PIVOT (<value_column> FOR <pivot_column> IN (<pivot_list>))`; the output pivoted table then is also a valid first-normal-form table. The text between the outermost parentheses constitutes the specification of the pivot operation. The first two columns in the pivot specification must be columns in the pivot operation's input table. These columns will not appear in the pivot operation's output table. Instead, each value in the pivot list within the pivot specification defines a new column in the pivot operation's output table. In the input table, elements in the pivot list appear as values in the pivot column. Corresponding values in the value column become values in the new columns in the output table. All columns of the input table not included in the pivot specification, called "grouping columns," are carried over to the output table.

In the example 400 set forth in Fig. 4, pivoting input table 410 in accordance with specification 420 produces output table 430. Pivot column 411, named `Quarter` in the input table 410 named `Narrow`, becomes four columns 431, 432, 433, and 434 in the output table 430. The names of these columns are the four distinct values, `Spring`, `Summer`, `Fall`, `Winter`, that appear as values in column 411, and that also appear in the value list following the keyword `IN` in 420. The sales numbers in value column 412 appear as values in corresponding ones of the four columns 431-434, but pivoted or rotated, so that sales figures for the same region and year are in the same row. Grouping columns 413-414 appear as columns 435-436 in output 430. In the output, the rows are grouped by equal values of the first grouping column 413, and then by equal values of the second grouping column 414, just as though specification 420 had contained an SQL clause of the form `GROUP BY Region, Year`. In this example, the effect of the pivot operation is to modify the perspective along which the data is viewed. Input table 410 presents data trends primarily by year for the Narrow regions of a company, whereas output table 430 allows seasonal tracking by quarters. (It should be recalled here that the rows

in an relational table do not have names, and have no particular order. Columns do have names and are sorted – that is, they are presented in the order their names appear in a query.) The pivot operation converts an input table having relatively many rows and relatively few columns into a result table having fewer rows and more columns.

The pivoted columns in the output table have the same data type (numeric, varchar, etc.) as the data in the value column of the input table. The value column, pivot column, and pivoted columns comprise simple data, rather than computed expressions. The order of the columns in both tables is not significant, as in ANSI SQL; columns can be referenced only by name, not by position. Although table 430 is shown sorted by values of grouping columns `Region` and `Year`, the pivot operation does not imply any particular sorting or ordering of the rows.

As mentioned above, a row in the input table does not appear in the output if its value does not appear in the pivot list. The input-table rows are grouped by equal values of any grouping columns, with respect to the definition of equality. Within each group, each row of the input table has a mutually distinct value in the pivot column. Each group results in one output row. For output columns not having a corresponding input row, the value is `NULL`, a special value defined in SQL.

Fig. 5 is a flowchart 500 of the steps carried out by modules 300, Fig. 3, of search engine 241 in Fig. 2 for a pivot operation. Block 510 receives a query from a user at a client terminal 120, Fig. 1, or from some other source as described herein. Step 520 identifies or selects which table 261 in database 260 is to serve as the input table of the operation. Step 521 identifies which column of the input table is to be the pivot column, step 522 identifies from the pivot list which pivot-column values participate in the pivot, and step 523 selects which column of the input is the value column. Step 530 constructs the output as another table 261. Step 531 emplaces a separate pivoted column for each data-item value in the pivot list. Step 532 constructs the grouping columns, if any. (As mentioned previously, these are any additional columns of the input table not identified in the pivot specification.) Step 540 inserts the data-item values of the value column into the rows of the output table as described previously; one method is by transposition, as indicated by step 541. Another way to express this transposition is that each data

item in the value column is placed into one of the pivoted columns, namely, that column whose name is the same as the data value in the pivot column of the input table. Step 550 groups the output-table rows by equal values of any grouping columns. Finally, step 560 stores the output table in database 260, Fig. 2.

The unpivot operation is the inverse of the pivot operation, and is formally defined as `<table_expression | query_expression>.UNPIVOT (<value_column> FOR <pivot_column> IN (<column_list>)`. The meanings of the terms are the same as for the pivot operation. Applying a pivot and an unpivot operation having the same specification to an input table restores the input table to its original state. In the example shown in Fig. 4, applying the unpivot operation 440 to pivoted table 430 restores table 410; the two named columns `Sales` 412 and `Quarter` 411 replace columns 431-434.

For each row in an input table, the unpivot operation generally produces one row of an output table per pivoted column. (However, a null value in a pivoted column does not generate an output row.) All columns in the pivot list must have the same data type in the input, and the entries in the value column of the output will have this type. Unpivoting a table increases the number of its rows and decreases the number of columns. Again, although Fig. 4 shows table 410 sorted by grouping-column values, the unpivot operation implies no row sorting in the output.

Fig. 6 is a flowchart 600 of the steps carried out by modules 300, Fig. 3, of search engine 241, Fig. 2 for an unpivot operation. Step 610 receives the unpivot operation and its specification. Because unpivot is defined to be the inverse of pivot, and to restore a pivoted table exactly to its unpivoted form, the specification of an unpivot is not complementary to that for a pivot, but rather has exactly the same form as that of the specification which created the pivoted table in the first place, as shown at 440 in Fig. 4. Again, step 610 may receive the operation from a user query or any other source. Step 620 identifies or selects the pivoted table such as 261 to be unpivoted. This table need not have been actually pivoted in a previous operation, but it normally will have been; that is, the pivot operation is generally used to achieve any initial rotation of perspectives, and unpivot is generally only used to restore a table to an

original unpivoted form in a clean, simple manner. Step 621 uses the specification's pivot list to identify which columns are the pivoted columns to be rotated or transposed. Steps 622 and 623 identify the names of the value column and the pivot column. These names are included in the specification because they do not appear anywhere within the pivoted table (at least, unless a previous pivot had kept a side table retaining this information).

Step 630 constructs the pivot table which is to be produced by the unpivot operation. Steps 631 and 632 form the pivot and value columns in the pivot table, using the names supplied in steps 622 and 623. Step 633 builds grouping columns in the pivot table, one for each of the pivoted-table columns not included in the unpivot specification received in step 610. Step 640 transposes the data items from the pivoted table into the unpivoted table constructed in the preceding steps. Names of the pivoted columns become data items in different rows, and the data items in the pivoted column go into the new value column, in the rows having respectively the same pivot-column values as the name of the pivoted column that they were in, in the original (pivoted) table. Step 650 groups rows by equal values of the grouping-column rows. Step 660 stores the table away in database 260, Fig. 2.

Correlation variables are not permitted within the specification of pivot and unpivot operations, because the pivot, value, and pivoted columns are simple data, and not computed values. These new operations have no bearing upon whether or which correlation variables are permissible in a query expression to which the operations are applied. If ANSI SQL permits defining table and column aliases for a query expression in which a pivot or unpivot operation does not occur, then it is acceptable to define such aliases for the query expression including a pivot/unpivot operation, but not for the query expression excluding the operation. For example, if `Table1 AS Table2 (col1, col2)` is permissible, then `Table1.PIVOT (...) AS Table2 (col1, col2, ...)` is acceptable, but `Table1 AS Table2 (col1, col2).PIVOT (...)` is not.

The pivot and unpivot operations may employ any number of conventional optimization techniques in block 330, Fig. 3. Because these operations are part of the relational-algebra level, an

algebraic query optimizer may be the most appropriate vehicle for realizing optimization techniques. Other optimization frameworks may be applicable as well.

Some additional optimization techniques may employ specific properties of the new operations. Obviously, a neighboring pair of pivot and unpivot operations may cancel each other, and may then be removed from a query. An optimizer should recognize that the grouping columns in the pivoted output table functionally determine the pivot and value columns, and therefore form a relational key of the result table. (This is very similar to the grouping columns in a conventional GROUP BY operation.) In an unpivot output table, the grouping columns together with the pivot column functionally determine the value column. These properties can assist in estimating the number of output rows for selectivity estimation and query-cost calculation for comparing alternative execution plans. They also may find utility in generating conditions for applying rewrite rules in simplifying the execution of a query. If a table is vertically partitioned, an operation to reassemble complete rows and a subsequent unpivot may cancel each other, eliminating both operations.

A conceptual similarity of the pivot operation to an SQL GROUP BY clause allows many techniques and rules for optimizing queries having that clause to serve as well for the new operations. Typical examples include: (1) pulling a pivot above a join, so as to reduce the grouping input's size, or to enable more efficient join algorithms; (2) pushing a pivot below a join to reduce the join input or to employ more efficient execution plans for the pivot; (3) merging two adjacent pivots, possibly effectively eliminating one of them; and (4) splitting a pivot into two parts, then pushing one of the parts through a join or across a process boundary, as a local/global aggregation in a parallel execution environment. In general, a query predicate on the grouping columns and a projection operation--including expressions that compute additional columns--can be moved through (either above or below) a pivot/unpivot operation in the same manner as a grouping operation.

Certain query predicates are more efficient to implement--and also easier to express--when treated as predicates (i.e., qualifications) against a pivot result table. For example, comparing two pivoted columns to each other is straightforward to express and efficient to implement, whereas the same

predicate applied to the pivot input table requires complex, inefficient nested queries. Therefore, rewriting the query to include a pivot/select/unpivot sequence of operations can be used to optimize such queries. For example, consider a query to select table rows in which `Sales` in the `Fall` exceeds `Sales` in the `Spring` in table 410, Fig. 4. Pivoted table 430 can accommodate this query as a comparison between the `Spring` and `Fall` columns 431 and 433, whereas the original table 410 requires joining the table to itself in order to perform the comparison. Using the tables of Fig. 4, such a larger query including pivot and unpivot operations (in conventional multiple-line format) might be:

```
(SELECT * FROM
Narrow.PIVOT (Sales FOR Quarter IN (Spring, Summer, Fall, Winter))
WHERE  Fall_Sales>Spring_Sales)
.UNPIVOT (Sales FOR Quarter IN (Spring, Summer, Fall, Winter))
```

Execution plans useful for block 340, Fig. 3, can be derived by those skilled in the art from conventional plans for grouping operations. In particular, plans based upon looping, indexing, streams, sorting, and hashing come readily to mind. Early-aggregation sorting and hybrid hashing are useful variants. Pivot/unpivot operations are amenable to parallel-execution environments, including parallel algorithms such as shared-memory, distributed-memory, shared-disk, and cluster machines. Local/global aggregation has already been mentioned as a possibility.

Unpivot operations require only a single-input, single-output plan that produces multiple output records for each input record. This operation can be easily executed in parallel on shared-memory, distributed-memory, shared-disk, and cluster machines.

Variations and Extensions

A number of variants and extensions to the above embodiment may be useful for some applications, either alone or in combination with each other. The most obvious, of course, are the replacement or augmentation of notational conventions, such as the dot invocation separator, and the rearrangement of components.

The pivot/unpivot operations as thus far described place limitations upon column names in the pivoted or output table. Column names in standard SQL must be character strings without spaces. Because column values may have spaces, and pivot changes values into names, a method can easily be devised to employ quoted identifiers and literals as column names. Likewise, it would be simple to represent column values having data types other than character strings as printable and readable representations for column names. Conventional name manipulations, such as concatenations of names might be useful. For example, a pivot list might contain column aliases using a keyword such as SQL `AS`. (In Fig. 4, if the `Quarter` values were "1" through "4" instead of season names, the specification of 420 might read `(Sales FOR Quarter IN (1 AS "Spring", 2 AS "Summer", 3 AS "Fall", 4 AS "Winter")`. In addition, `AS` might be used to rename pivot-result columns in any context, and user-defined functions could be supplied for converting complex column names, or for conforming names to specific limitations of an SQL implementation.

Semantic extensions might include pivoting and unpivoting multiple columns in a single step. For example, operation 450 in Fig. 4 replaces the three columns 411, 412, and 414 with eight columns, instead of the four columns 431-434 of result table 430. That is, the set of pivot columns represent a Cartesian or outer product of all the pivot lists. A convention for naming the pivoted columns might merely involve concatenating the names, such as `Sales_1996_Spring`, etc. A multicolumn unpivot operation with the same specification could decode such names back into their original form, so as to provide a true inverse for this extension. A further extension would permit multicolumn pivoting in steps. For example, it might be desired to apply a further pivot to already pivoted table 430 about column 436, to produce the eight columns described above. Rather than unpivoting table 430 and then applying a multicolumn pivot, an extended form allows a list of columns in place of the value column, e.g., `Wide.PIVOT ((Spring, Summer, Fall, Winter) FOR Year IN (1996, 1997))`. Optimizer 340 and compiler 350 can easily collapse these operations into a single plan for execution.

The pivot operation (but not unpivot) can support conventional SQL aggregation or grouping functions such as `MIN`, `SUM`, `AVG`, and even `COUNT`, for the value column at step 540. In that case, the limitation to a single row per group can be lifted. Of course, the type of the new column might differ from that of the original. The following example, based upon Fig. 4, illustrates a query using aggregation:

```
(SELECT Year, Quarter, Sales FROM Narrow)
.PIVOT (SUM(Sales) FOR Quarter IN (Spring, Summer, Fall, Winter)
```

This query consolidates `Sales` for the `East` and `West` regions into a single sum representing the entire company for each `Year`. In versions where grouping functions are allowed, the implementation could specify the implicit application of a particular function, such as `SUM`, in all cases where a pivot operation would otherwise produce duplicate primary keys in different rows of the pivoted table. Pivots with grouping cannot be reversed, because the aggregation loses information detail; grouped output in standard SQL cannot be reversed for the same reason. Although this extension prevents unpivot from functioning as a true inverse, an embodiment preserves this capability by adding an internal "side table" that saves all the original values.

Another powerful extension adds a capability for replacing the list of literal column names in a pivot or unpivot operation with a `SELECT` query. More complex processing would involve running the auxiliary query first, then binding the list of pivoted columns using the result of the auxiliary query; that is, the auxiliary query requires interleaved compilation and execution in blocks 340 and 350, Fig. 3. The execution requires computing the query expression that is to be pivoted, as well as running a query against the result.

The pivot specification might omit the pivot list entirely, supplying a default query instead. For example, omitting the clause `IN (Spring, Summer, Fall, Winter)` from query 420 could substitute a default query `SELECT DISTINCT Quarter FROM Narrow`. Because this causes query 420 to reference the input table twice, it would be useful to introduce a dedicated name for an operation's input table, analogously to the name "this" in C++. Operation 420 would then become:

```
Narrow.PIVOT (Sales FOR SELECT DISTINCT Quarter FROM INPUT).
```

Instead of requiring a pivot list, the unpivot operation might allow a specification of all but the pivoted columns in the operation's input. In the example 400 as modified above, the inverse operation could be specified as (see 440, Fig. 4:

```
Wide.UNPIVOT (Sales FOR Quarter IN (Spring, Summer, Fall, Winter))
```

or as (460, Fig. 4):

```
Wide.UNPIVOT (Sales OVER (Region, Year)).
```

Supporting OVER in this context necessitates determining the set of pivoted columns from the input table, and thus requires the ability to process auxiliary queries, as described above. The IN and OVER clauses can be combined, permitting one or more columns to be a pivoted column as well as a grouping column. A situation where this might make sense from the application's perspective is the inclusion of Spring sales in each output row, in order to allow computation of sales growth since the first quarter for each subsequent quarter.

In some tables, a set of columns can be more or less orthogonal or independent; for example, columns named "City" and "Month" are likely to have table entries for all cities for all months. Other column sets are hierarchical--such a "Locations" table having "State", "City", and "Store" columns--and their data is sparse; that is, very few cities will occur in multiple states, and few cities will have multiple stores. In the latter case, the use of two IN clauses leads to ungainly syntax and semantics in a pivot operation. However, employing a list of pivot columns instead of a single pivot column ameliorates this problem. ANSI SQL's concept of "row values" is appropriate to this case. Typically, although not always, it is more convenient to specify a query as the pivot list, rather than a list of literal column names. An exemplary form might be `Locations.PIVOT (SalesVolume FOR (City, Store) IN (SELECT City, Store FROM Outlets))`.

The pivot and unpivot operations could also find utility in the internal operation of query processors 300, Fig. 3 In addition, referential-integrity constraints need to be enforced only for deleted candidate keys and new foreign keys; using pivot/unpivot or a similar rotation could collapse deletion and

insertion items pertaining to the same key value, and thus potentially eliminate some integrity checks as redundant. Moreover, in a conventional query having a very large `IN` clause, an internal unpivot operation invoked implicitly by the search engine can map a single very complex row containing many literals or parameters as columns into a set of rows that can be matched against database tables, using conventional join methods such as loops, index, merge, and hash join, and their parallel-processing variants. Similar internal invocations of pivot/unpivot operations may be useful in `OR` and `UNION` queries, which are often equivalent to `IN` clauses.

Finally, the pivot and unpivot operations, along with their extensions, optimizations, and execution plans, can be included in database management systems and data-manipulation software outside the field of relational database systems. For example, an algebra-based statistical-analysis product or a standalone sorting package may find these new operations useful, both externally and internally. Although described in connection with the SQL language, the operations need not necessarily be incorporated into SQL, or indeed into any host language.

Claims

1. A method of pivoting data from an input table to form an output table in a relational database management system on a digital computer, comprising:

identifying the input table;

identifying a pivot column of the input table;

identifying a pivot list of data items from the pivot column;

identifying a value column of the input table; and

transposing the input table about the pivot column based upon the data items in the pivot list to construct the output table.

2. A method according to claim 1, comprising the further steps of:

identifying at least one grouping column of the input table; and

grouping the rows of the output table according to equal values of the data items in the one or more grouping columns.

3. A method according to claim 1, wherein the transposing step occurs at a central server responding to users located at multiple remote client locations, and wherein the identifying steps originate with one of the users.

4. A method according to claim 1, wherein the identifying steps originate in a search engine at the central server as a part of optimizing a query from a user which query does not include the pivot method as an explicit operation.

5. A method of transforming data from an input relational database table stored in an electronic data processor into an output table, both of the tables having an array of data values organized as a plurality of

rows and columns and having multiple column names separate from the data values and associated with respective ones of the columns, the method comprising:

selecting the name of a first of the columns of the input table as a pivot column;

selecting the name of a second of the columns of the input table as a value column;

accessing the input table in the data processor;

converting a set of the data values in the pivot column into respective column names associated with multiple pivoted columns in the output table;

for each data value in the value column located in the same row of the input table as a particular one of the set of data values in the pivot column, placing the particular one data value into a certain one of the pivoted columns of the output table, the certain one pivoted column having a name corresponding to the particular one data value in the pivot column; and

storing the output table in the data processor.

6. A method according to claim 5, comprising the further step of selecting certain ones of the set of data values in the pivot column as a pivot list, and wherein said converting step converts only those data values in the pivot list into the column names.

7. A method according to claim 6, wherein the converting step does not convert into column names any rows in the input table having NULL data values in the pivot column.

8. A method according to claim 5, wherein the input table includes columns other than the pivot and value columns, the method comprising the further step of grouping the rows of the output table by equal values of the data items in at least some of the other columns.

9. A method according to claim 8, wherein the grouping step groups the rows of the output table by equal values of the data items in all of the other columns.

10. A method of unpivoting data from a pivoted table to form an unpivoted table in a relational database management system on a digital computer, comprising:

identifying the pivoted table;

identifying a pivot-column name for the unpivoted table;

identifying a pivot list of columns of the pivoted table;

identifying a value-column name for the unpivoted table;

constructing a pivot column and a value column in the unpivoted table; and

transposing the pivoted table about the columns in the pivot list so as to place the names of the pivot-list columns as data items in the pivot column of the unpivoted table, and to place data items in the pivot-list columns into rows of the value column of the unpivoted table.

11. A method according to claim 10, comprising the further steps of :

identifying at least one grouping column of the pivoted table; and

grouping the rows of the unpivoted table according to equal values of the data items in the one or more grouping columns.

12. A method according to claim 10, wherein the transposing step occurs at a central server responding to users located multiple remote client locations, and wherein the identifying steps originate with one of the users.

13. A method according to claim 10, wherein the identifying steps originate in a search engine at the central server as a part of optimizing a query from a user which query does not include the pivot method as an explicit operation.

14. A method of transforming data from a pivoted relational database table stored in an electronic data processor into an unpivoted table, both of the tables having an array of data values organized as a plurality of rows and columns and having multiple column names separate from the data values and associated with respective ones of the columns, the method comprising:

selecting a name as the name of a pivot column;

selecting a plurality of names of the columns in the pivoted table as a pivot list;

selecting a name as the name of a value column;

accessing the pivoted table in the data processor;

creating a pivot column in the unpivoted table, having a name selected as the pivot column name;

converting the column names in the pivot list into data values in the pivot column;

creating a value column in the unpivoted table having the name selected for the value column;

for each particular data value in each particular column of the pivoted-table columns in the pivot list, placing the particular data value into the value column of the unpivoted table in a row which also contains a data value in the pivot column corresponding to the particular column of the pivoted table; and

storing the unpivoted table in the data processor.

15. A method according to claim 14, wherein the pivoted table includes columns other than the columns in the pivot list, the method comprising the further step of grouping the rows of the unpivoted table by equal values of the data items in at least some of the other columns.

16. A method according to claim 15, wherein the grouping step groups the rows of the unpivoted table by equal values of the data items in all of the other columns.

17. A relational database system, comprising:

a number of clients; and

a search engine including modules for parsing, optimizing, and executing a query from one of the clients containing a pivoting operation specifying an input table in a relational database, a name of a pivot column and of a value column in the input table, and a pivot list of data values in the pivot column, the search engine transposing data items in the value column of the input table about the pivot column based upon the data items in the pivot list so as to construct a pivoted output table.

18. A system according to claim 17, wherein the search engine reside in a central server, and the clients are physical located at multiple locations remote from the server.

19. A system according to claim 17, wherein the optimizer module optimizes the query including the pivot operation.

20. A system according to claim 17, wherein the query from the one client includes an unpivot operation specifying an unpivoted table for inverting the effect of the pivot operation upon the pivoted table by transposing data items in the pivoted table about a number of pivoted columns.

21. A system according to claim 20, wherein the data items in the pivoted column are placed into the value column of the unpivoted table.

22. A pivoted table in a relational data base, the pivoted table having data values placed in rows and columns according to a pivot specification involving an input table, a pivot column, and a value column, the pivoted table comprising:

a plurality of pivoted columns each having a column name corresponding to a distinct one of the data values in the pivot column; and

a plurality of rows each containing multiple data items from the value column, each of the data items in one row being located in that one of the pivoted columns whose name corresponds to a row of the input table containing the name of the one pivoted column as a data value.

23. A table according to claim 22, wherein the specification also includes a pivot list of pivot values, and wherein the pivoted columns of the output table consist of those columns corresponding to data values in the pivot column appearing in the pivot list.

24. A table according to claim 22, wherein the input table contains a further column in addition to the pivot and value columns, and wherein the output table further comprises a grouping column containing data values from the further column of the input table.

25. A table according to claim 24, wherein the rows of the output table are grouped by equal values of the data items of the grouping column.

26. A table according to claim 25, wherein the input table contains a plurality of the further columns, each of the further columns producing a separate grouping column in the output table.

27. A table according to claim 22, wherein the table is stored on a storage medium.

28. An unpivoted table in a relational database, the unpivoted table having data values placed in rows and columns according to a pivot specification involving a pivot input table, a pivot column, a pivot list, and a value column, the unpivot table comprising:

a pivot column having a name corresponding to the pivot column, and whose rows contain data items derived from the names of pivoted-table columns appearing in the pivot list; and

a value column having a name derived from the value column in the specification, each particular row of the value column containing a data item derived from the data item in that one of the pivoted columns whose name is a data item in the same particular row of the unpivot table.

29. A table according to claim 28, wherein the pivoted table contains at least one further column in addition to the pivoted columns.

30. A table according to claim 29, wherein the rows of the unpivoted table are grouped by equal values of the at least one further column.

31. A table according to claim 28, wherein the table is stored on a storage medium.

32. A data-storage medium having a program stored thereon for causing a suitably programmed computer to perform the steps comprising:

receiving a specification identifying an input table, a pivot column of the input table, a pivot list of data items from the pivot column, and a value column of the input table; and

transposing the input table about the pivot column based upon the data items in the pivot list to construct an output table.

33. A data-storage medium according to claim 32, wherein said programmed computer performs the further steps of :

defining at least one grouping column of the input table; and

grouping the rows of the output table according to equal values of the data items in the one or more grouping columns.

34. A data-storage medium having a program stored thereon for causing a suitably programmed computer to perform the steps comprising:

identifying the pivoted table, a pivot list of columns from the pivoted table, and a pivot-column name and a value-column name for an unpivoted table;

constructing a pivot column and a value column in the unpivoted table; and

transposing the pivoted table about the columns in the pivot list to place the names of the pivot-list columns as data items in the pivot column of the unpivoted table, and to place data items in the pivot-list columns into rows of the value column of the unpivoted table.

35. A data-storage medium according to claim 34, wherein said programmed computer performs the further steps of :

identifying at least one grouping column of the pivoted table; and

grouping the rows of the unpivoted table according to equal values of the data items in the one or more grouping columns.

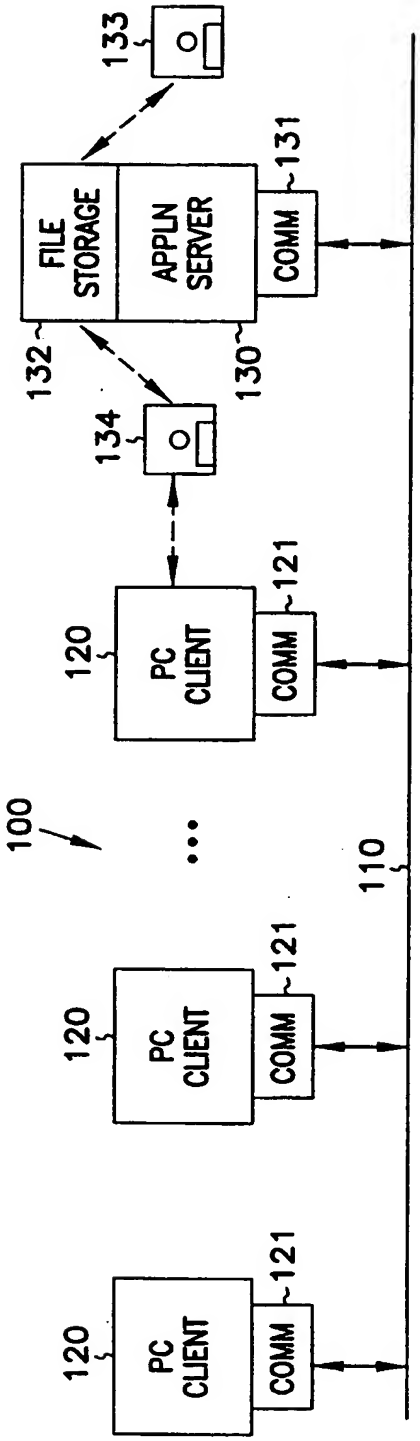

133
FILE STORAGE
APPLN SERVER
COMM
131
132
130
134
120
PC CLIENT
121
100
110

FIG. 1

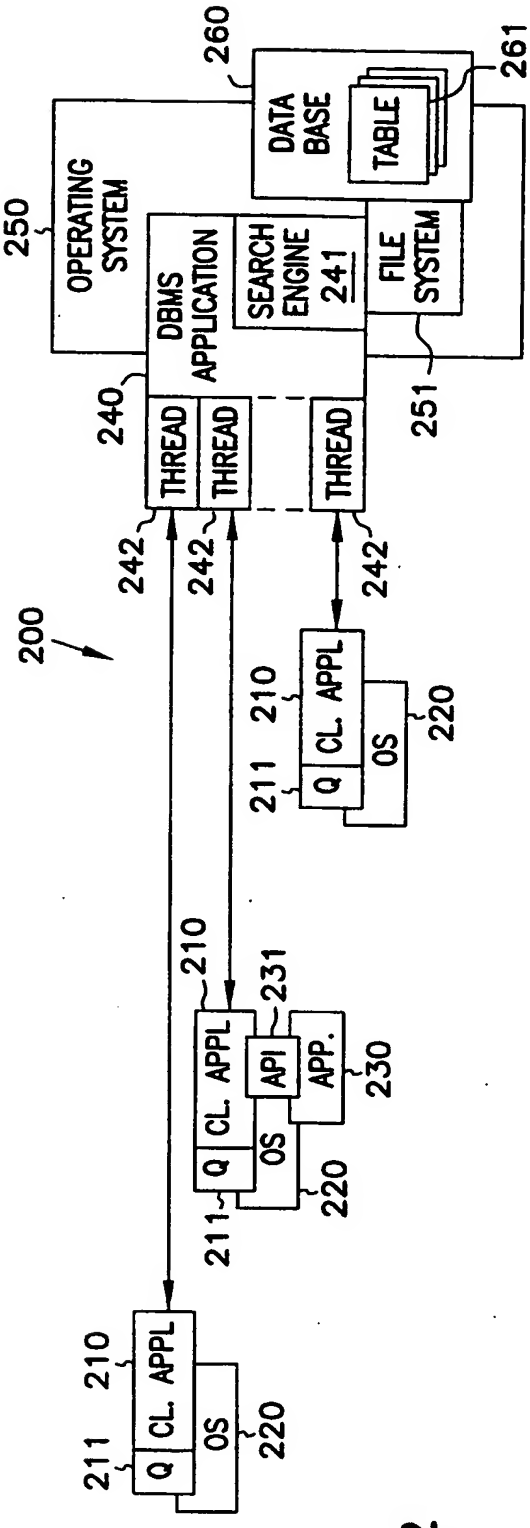

260
DATA BASE
TABLE
261
250
OPERATING SYSTEM
DBMS APPLICATION
SEARCH ENGINE 241
FILE SYSTEM
240
THREAD
242
251
200
210
CL. APPL
211
Q
OS
220
231
API
APP.
230

FIG. 2

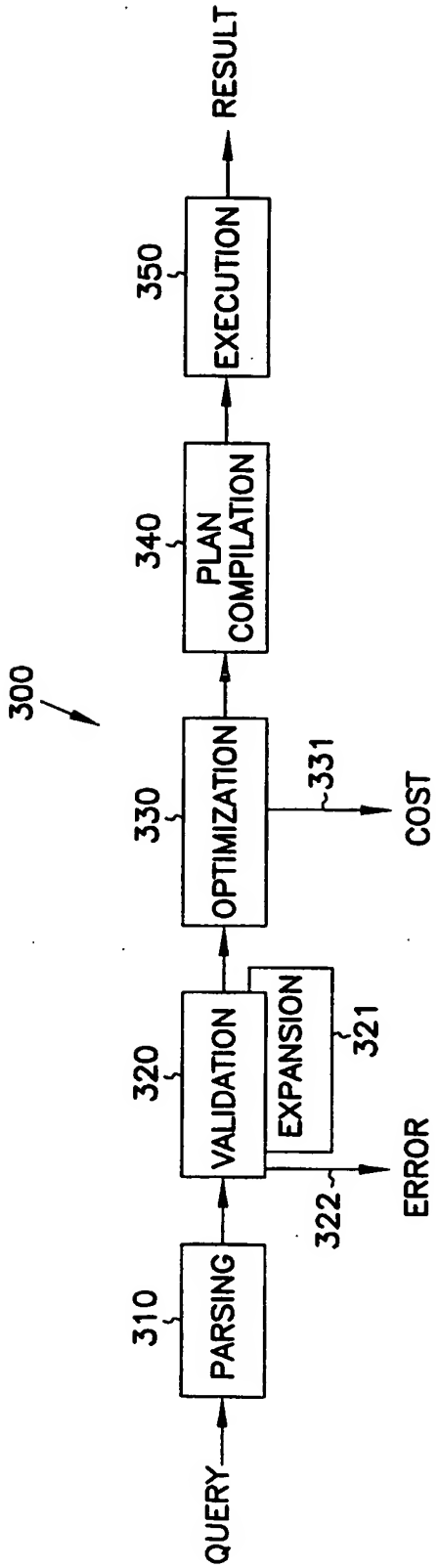

300
QUERY
310
PARSING
320
VALIDATION
EXPANSION
321
322
ERROR
330
OPTIMIZATION
331
COST
340
PLAN COMPILATION
350
EXECUTION
RESULT

FIG. 3

400

410

NARROW

413 REGION	414 YEAR	411 QUARTER	412 SALES
EAST	1996	SPRING	49000
EAST	1996	SUMMER	51000
EAST	1996	FALL	51000
EAST	1996	WINTER	64000
EAST	1997	SPRING	52000
EAST	1997	SUMMER	54000
EAST	1997	FALL	53000
EAST	1997	WINTER	66000
WEST	1996	SPRING	30000
WEST	1996	SUMMER	31000
WEST	1996	FALL	30000
WEST	1996	WINTER	36000
WEST	1997	SPRING	29000
WEST	1997	SUMMER	29000
WEST	1997	FALL	32000
WEST	1997	WINTER	37000

430

WIDE

435 REGION	436 YEAR	431 SPRING	432 SUMMER	433 FALL	434 WINTER
EAST	1996	49000	51000	51000	64000
EAST	1997	52000	54000	53000	66000
WEST	1996	30000	31000	30000	36000
WEST	1997	29000	29000	32000	37000

NARROW.PIVOT (SALES FOR QUARTER IN (SPRING, SUMMER, FALL, WINTER)) 420

WIDE.UNPIVOT (SALES FOR QUARTER IN (SPRING, SUMMER, FALL, WINTER)) 440

NARROW.PIVOT (SALES FOR YEAR IN (1996, 1997) AND QUARTER IN (SPRING, SUMMER, FALL, WINTER)) 450

WIDE.UNPIVOT (SALES OVER (REGION, YEAR)) 460

FIG. 4

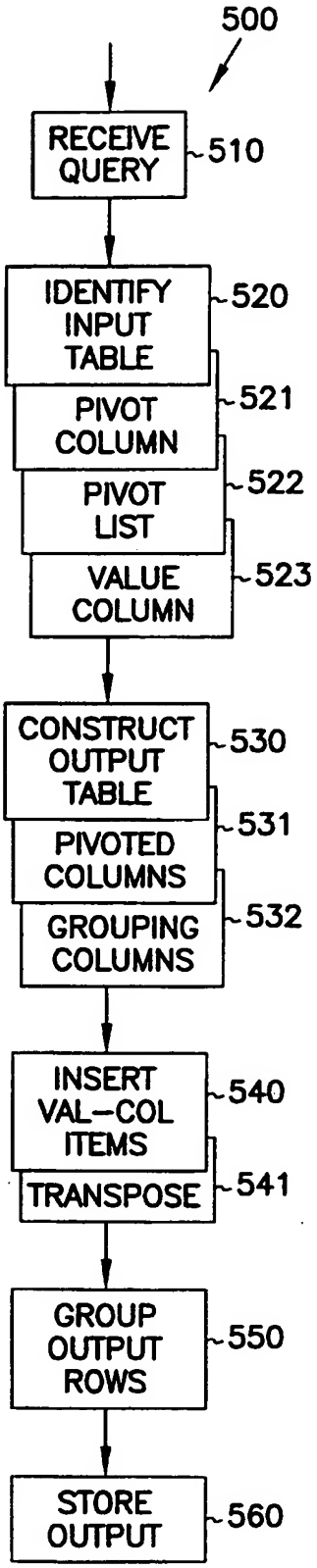

500
RECEIVE QUERY
510
IDENTIFY INPUT TABLE
520
PIVOT COLUMN
521
PIVOT LIST
522
VALUE COLUMN
523
CONSTRUCT OUTPUT TABLE
530
PIVOTED COLUMNS
531
GROUPING COLUMNS
532
INSERT VAL-COL ITEMS
540
TRANSPOSE
541
GROUP OUTPUT ROWS
550
STORE OUTPUT
560

FIG. 5

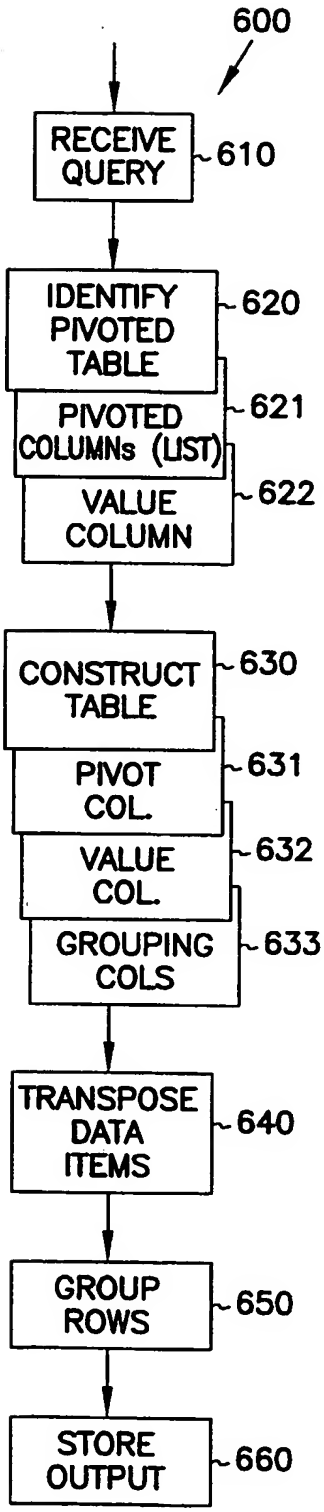

600
RECEIVE QUERY
610
IDENTIFY PIVOTED TABLE
620
PIVOTED COLUMNs (LIST)
621
VALUE COLUMN
622
CONSTRUCT TABLE
630
PIVOT COL.
631
VALUE COL.
632
GROUPING COLS
633
TRANSPOSE DATA ITEMS
640
GROUP ROWS
650
STORE OUTPUT
660

FIG. 6

INTERNATIONAL SEARCH REPORT

Inter onal Application No
PCT/US 99/05716

A. CLASSIFICATION OF SUBJECT MATTER
IPC 6 G06F17/30

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
IPC 6 G06F

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practical, search terms used)

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category °	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	J. C. NOSSITER: "Using Excel 5 for Windows" 1995 , QUE CORP. , USA XP002109365 page 256, line 1 - page 258, line 1 page 276, line 15 - page 281, line 3	1,5,10, 14,17, 22,28, 32,34.
A	WO 95 11487 A (FDC INC) 27 April 1995 (1995-04-27) page 1, line 1 - page 4, line 30 page 6, line 21 - page 13, line 23; figures 1,3	1,3-5, 10, 12-14, 17-19, 22,28, 32,34

☐ Further documents are listed in the continuation of box C.

☒ Patent family members are listed in annex.

° Special categories of cited documents :

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier document but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.

"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
15 July 1999	17/08/1999

Name and mailing address of the ISA	Authorized officer
European Patent Office, P.B. 5818 Patentlaan 2 NL - 2280 HV Rijswijk Tel. (+31-70) 340-2040, Tx. 31 651 epo nl, Fax: (+31-70) 340-3016	Fournier, C

INTERNATIONAL SEARCH REPORT

Information on patent family members

Inter nal Application No
PCT/US 99/05716

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
WO 9511487 A	27-04-1995	GB 2298941 A,B	18-09-1996
		US 5845276 A	01-12-1998